

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
22 August 2002 (22.08.2002)

PCT

(10) International Publication Number
WO 02/065286 A2

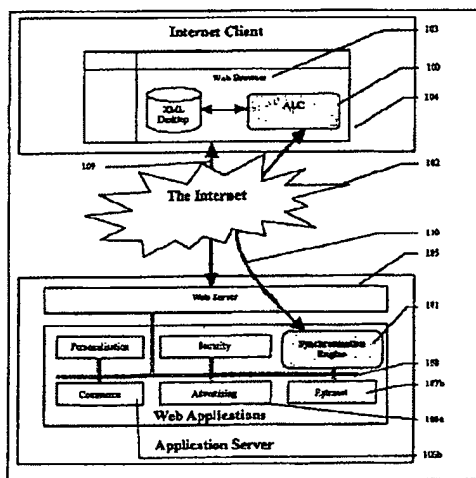
- (51) International Patent Classification⁷: **G06F 9/46** (74) Agent: **LANGLEY, Peter, James**; Origin Limited, 52 Muswell Hill Road, London N10 3JR (GB).
- (21) International Application Number: **PCT/GB02/00578**
- (22) International Filing Date: 12 February 2002 (12.02.2002) (81) Designated States (*national*): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:
0103308.3 12 February 2001 (12.02.2001) GB
0112433.8 22 May 2001 (22.05.2001) GB
- (71) Applicant (*for all designated States except US*): **ALTIO LIMITED** [GB/GB]; Adelaide House, 626 Chiswick High Road, London W4 5RY (GB).
- (72) Inventors; and
(75) Inventors/Applicants (*for US only*): **LEVETT, David, Lawrence** [GB/GB]; 52 Arethusa Way, Bisley, Woking, Surrey GU24 9BX (GB). **MILLS, Robert, Ian** [GB/GB]; 9 Southern Hay, Hartley Wintney, Hampshire RG27 8TZ (GB). **NATHAN, Benjamin, Lawrence** [US/GB]; 54 Ravenscroft Avenue, London NW11 8AU (GB).
- (84) Designated States (*regional*): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— without international search report and to be republished upon receipt of that report

[Continued on next page]

(54) Title: CLIENT SOFTWARE ENABLING A CLIENT TO RUN A NETWORK BASED APPLICATION



(57) Abstract: Client software, enabling a client to run a network based application which uses structured data, in which the client software comprises: (a) a communication layer to send and receive messages over the network; (b) a database layer to store, and allow querying of, the structured data; (c) a rendering layer which generates, from the structured data in the database layer, data for a user interface; wherein the client software is self-contained to provide all of the communications, data storage and rendering resources needed to run the network based application on the client device. The 3 Layer System is fully integrated and therefore requires no additional client side code to be written. Normally, this level of self-containment on the client side does not exist, so that a developer wishing to deploy a network based application to a client device needs to write client side custom code for the user interface.

WO 02/065286 A2

WO 02/065286 A2



For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

CLIENT SOFTWARE ENABLING A CLIENT TO RUN A NETWORK BASED APPLICATION

5 BACKGROUND TO THE INVENTION

1. Field of the invention

This invention relates to client software enabling a client to run a network based application.
10 It finds particular utility in enterprise computing. A network based application is any application in which data has to be sent to a client device over a network, such as a WAN like the internet or a LAN. It includes web applications, back end applications and web services.

15 2 Description of the Prior Art

Enterprise computing was originally conducted through large mainframe computers. These mainframes gave the end-user very limited application functionality, displaying only text messages and requiring keyboard strokes for every command (for example, there was no
20 mouse and none of the richness of today's desktop applications like drag and drop, graphing, sorting or personalized settings). Deployment costs were also very high due to the cost of hardware and the lack of skilled computer operators.

The evolution to client/server-based computing provided much richer functionality. The
25 graphical user interface (GUI) provided end-users a much simpler, more efficient method to interact with applications, but systems still proved expensive and difficult to maintain. In theory, applications would be deployed on the server centrally and distributed via the network to each of the clients. But business users had different needs of functionality and therefore required different applications with unique configurations. Not only did some

information have to be personalized, but information also had to be retained on the desktop. This meant it was not possible to “write once and deploy many”. The result was that the end-user got the desired rich functionality, but much of the code for the business logic for the applications was located on individual clients. IT Departments’ budgets ballooned due to high support costs and slow implementation times. Rolling out new applications – or even minor updates – required a unique installation for every PC that required access.

The advent of the Internet was supposed to change all this. The Internet provided an infrastructure that lent itself to the notion of zero-cost deployment with the promise of “write once, run anywhere”. Such a model would significantly decrease the cost of deploying enterprise systems. With the Internet, back-end servers would run the application centrally, and all that users required to access the application was a browser. There would be no code specific to the application running on the user’s client. The potential this promised to large companies was unprecedented.

Despite the enormous potential, the initial technology (HTML) used for building and deploying the front-end of these new Internet applications was never intended for anything more than displaying simple text. As companies tried to deploy sophisticated business systems over the Internet, they soon realized that much of the desired functionality they wanted was not feasible. HTML based Web Applications did not have true interactivity or access to data in real-time. Business users needed to interact with and manipulate the data, but mostly what they saw was static page views. Certain solutions were designed to provide a more interactive user interface. However, they still used variations of HTML, resulting in passive documents with numerous limitations.

Specifically, from a technical perspective these limitations resulted in:

- **Lack of functionality:** No matter how sophisticated the back-end business logic, end-users lost the functionality they had come to expect from their familiar client/server-based

system. Large online systems that were deployed severely lacked the rich functionality that companies demanded in their offline systems. While lots of money was spent on building these systems, to date they have clearly not replaced client/server systems. In fact, very few even augment the existing offline systems, and in retrospect proved to be very expensive IT exercises which fell short of their original promise. There have been early attempts at providing a richer user interface and a more robust connection between the back-end servers and the client by using JavaScript, DHTML, and various plug-ins. However, these solutions present expensive development cycles and ongoing maintenance, while introducing new security issues (see below), and applications that still fall short of the desired functionality.

• **Load on back-end Servers:** The HTML-based Web represents a very inefficient use of computing resources. Publishing HTML's required page by page structure means that as more people log on to a system, requests to the database and the application server put a strain on these servers. Companies needing to scale operations can buy more servers with more CPUs to handle the high load, but each additional processor not only costs a lot, but may also affect the license cost of all the various software packages running on the server.

• **Security Concerns:** Companies are concerned that data pushed beyond corporate firewalls onto the Internet is open to unauthorized, third-party scrutiny. Additionally, there is concern that plug-ins downloaded from various Internet applications may contain viruses. Most software companies license their software on a per-CPU basis. As companies add more CPUs to their servers or add entirely new servers, the licensing fee for each piece of software increases by a multiple of how many new processors are added.

• **Cannot Easily Prototype New Features:** Given the complexity of developing user interfaces that provide adequate functionality and the effort required to integrate such solutions to the back-end, companies typically do not prototype and roll out new features incrementally. Very often, further development of these systems is an extensive project, which introduces the uncertainty of never being fully deployed and used. Because of the

effort required for each release, many companies shy away entirely from going forward on certain projects as opposed to rapidly developing incremental pieces to see what parts work.

From a business manager's perspective, these limitations meant

5

• **Inadequate Systems for Doing Business:** Client/server-based systems were expensive, difficult to build and deploy and not accessible outside of the enterprise. However, they do provide the functionality needed to actually do business, whether for trading financial instruments, managing resources and accounts through ERP systems or selling and maintaining client relationships through SPA and CRM tools. A fraction of these capabilities are available over the Internet in certain systems, but are far from giving the functionality people require to be effective.

15 • **Slow Performance:** Sites using HTML force users to click and wait for a response while the back-end system processes the information and pushes a new page to the users. This can take a few seconds at best, or tens of seconds, even minutes, at worst. Users of these systems simply abandon activities when they take too long or when connections entirely fail.

20 • **Expensive Development:** Building and deploying enterprise-level systems over the Internet with sophisticated functionality requires a lot of custom coding with long and expensive development cycles. Most of this work cannot be leveraged for other systems and is a large, one-off expense that companies must absorb.

25 • **Expensive Maintenance:** Complex Web systems contain thousands of lines of code on both the back-end (business logic) and the front-end (delivers the interface). The connection between the front-end and backend also contains thousands of lines of custom code which needs to be debugged and maintained. All changes to the code require programmers to manually write new code and test the functionality through staging before going to a live production site.

• **Expensive Bandwidth:** In today's online systems, information is presented to users through entire Web pages, which are pushed out to users over the network. As information changes or transactions are completed, users can request new pages or submit pages back to the system. However, every time pages are sent across the network, companies pay for the bandwidth used. As online systems add users and these users start to do more activities online, they consume more and more bandwidth, and as a result costs increase quickly.

• **Mobile Access:** As the reach of the Internet extends beyond the PC, enabling mobile access (i.e. PDAs, Cellular Phones) to business applications is becoming a standard business requirement. Current solutions for wireless delivery require separate application re-writes (and expertise) for each device supported.

• **Hiring and Retraining expensive IT Staff:** Integration of Web systems with existing offline systems and the maintenance of more sophisticated user interfaces for the Web using JavaScript and HTML typically requires armies of programmers, burdening the budget of every department, and many times putting the overall company's viability in jeopardy.

As a result of these challenges experienced to date, companies still have limited confidence in deploying their systems through the Web. Re-creating the same feature-rich interfaces that existed on client/server-based desktop applications has proven futile for companies attempting to move their systems to the Internet. Replicating the performance of desktop applications as well as standardizing the collaboration and interaction between disparate Web applications has been even more complex, expensive, and mostly elusive. The massive trade-off in end-user productivity, combined with security, scalability and performance concerns, has resulted in enterprises running client/server systems in parallel with Web-enabled systems, further deepening the cost of IT, and still not realizing most of the benefits the Internet promised.

Hence, even though millions of people are now using the Internet to look at documents from anywhere in the world, true interactivity to manipulate data is missing, and the evolution of the Web into a mainstream business and trading environment simply has not happened.

SUMMARY OF THE PRESENT INVENTION

In a first aspect of the invention, there is provided:

- Client software, enabling a client to run a network based application which uses structured data, in which the client software comprises:
- (a) a communications layer to send and receive messages over the network;
 - (b) a database layer to store, and allow querying of, the structured data;
 - (c) a rendering layer which generates, from the structured data in the database layer, data for a user interface;
- wherein the client software is self-contained to provide all of the communications, data storage and rendering resources needed to run the network based application on the client device.

- Prior art approaches to solving the problem of deploying network based applications to clients (e.g. interactive web based applications) suffer from the extensive problems identified in the preceding section. For example, they have consistently required programmers to write client based code: Internet Explorer™ from Microsoft Corp, Redmond, USA, includes an XML Document Object Model (DOM) – i.e. an XML database which will run on the client device. But a programmer wishing to deploy a web based application to a client running the Internet Explorer DOM needs to craft JavaScript or other ‘code’ to interpret data in the XML DOM and effect changes to a DHTML page in the browser. Alternatively, a programmer might use a tool to automatically generate and embed the ‘code’ into the DHTML page. In either case, the code is explicitly required to generate the conversion from XML to user interface. No such client side software development is needed with the present invention because it is a self-contained and complete ‘3 Layer System’ providing comprehensive communications/database/renderer functions.

The 3 Layer System is fully integrated and therefore requires no additional client side code to be written. As data (‘Application Data’) changes in the Database Layer, it automatically

communicates with the Rendering Layer causing it to update the user interface appropriately. Similarly, any changes to the user interface caused by user interaction (eg pushing a button, turning a dial or entering in new information through a keyboard) can cause the Rendering Layer to notify the Database Layer of any changes that have taken place.

5

Changes to the Database layer caused by the Rendering Layer can automatically cause the Database Layer to contact the Communications Layer and can cause it in turn to send messages over a network that indicate modifications to the Application Data on the client.

- 10 This seamless, integrated and automatic interaction between these layers in both directions can be done without the need for traditional programming methods and is inherent to core implementations of the invention.

The result is that this invention enables the rapid creation and deployment of highly
15 interactive, network aware applications whilst taking full advantage of the rich structure within the application's data, and without the need or expense of traditional programming approaches.

Interaction between the client and the server is carried out at the Application Data level
20 between the server and the Database Layer via the Communications Layer. The server need not even know of the existence of a Rendering Layer on the client. Thus, the 3 Layer System serves to fully insulate the user interface of the client device from a server hosting the network based application. Normally, this level of self-containment on the client side does not exist, so that a developer wishing to deploy a network based application to a client device
25 needs to write client side custom code for the user interface, as in the Internet Explorer example given above. Also, in the traditional example, the server will often know about the rendering process on the client, and in most cases will generate the combination of Application Data and rendering instructions needed to render the information on the user interface.

The client software needs some limited hooks into standard resources on the client, namely those necessary for it to control the user interface of the client device (e.g. a screen or a line of LEDs), or provide non-visual functions (the term 'rendering' should not be construed as implying only visual rendering, but to controlling any kind of device interface). It needs
5 hooks into standard resources to send and receive data over the network. In one implementation, the client software relies on a Java Virtual Machine for this, but it can equally be run natively on an operating system; because it is self-contained (i.e. contains its own communications manager, database and rendering system), porting to run on different operating systems is straightforward.

10

By tightly integrating the major infrastructure components needed to support a Network Based Application, implementations of the invention will often be significantly smaller in size and more efficient than were the components to be independent. As a result, these implementations are suited to run on devices with limited memory and processing resources,
15 and/or be deployed automatically at run-time over a network.

In one implementation, the database layer holds, independently of the structured Application Data, 'Configuration Data' which defines how that data can be interacted with from within a user interface. Hence, this database is not just a record set (i.e. the product generated by a
20 database), nor is it simple HTML, in which data and configuration data are not independent but are, conversely, inseparable. Independence leads to a key advantage: the rendering layer continuously combines the Configuration Data and the structured Application Data (or rather sub-sets of both) at display time; Configuration Data can therefore be changed asynchronously from any user interaction. This independence of the Application Data and
25 the Configuration Data enables the server to which the networked client is connected, to interact separately for changes to the Application Data and Configuration Data, and in many cases restrict its communications with the client to changes to the Application Data alone.

In one implementation, the structured Application Data and the Configuration Data is provided by a server (usually, but not invariably, physically remote from the client; a server on the same device as the client is a possibility: it could then be a closed system). Normally, the user interface of a network based application (e.g. a web based application) is defined by the client – i.e. the look and feel of the data and the data displayed at the client is defined by the client. But in this implementation, it is the server which defines this user interface; both Application Data and Configuration Data on the client database layer can be updated in real time by the server.

- 10 Changes and additions can be made to the Application Data through user interactions with the Rendering Layer as an individual change (e.g. pushing a button), or a collection of changes (e.g. filling in a form or setting a group of switches).

15 Changes to the Application Data can be configured to apply locally (i.e. not cause an update via the Communications Layer to the server). This might be used to store temporary changes, (e.g. should the user wish to pre-fill in a form but not submit it until some other event has happened).

20 When Application Data is added or modified at a client device (e.g. a new customer name is added to a list of customers), it is possible to enforce that the new name is displayed on the client device only once the server has processed a request from the client to add the new customer, has verified that the change is allowed and has sent updating data to the client. The updating of the user interface (e.g. to show the new name) can also be independent of the submission of the initial request from the client to the server (to add the new customer).

25 In this way, the user who initiated the change (e.g. entered the new customer name) receives rapid feedback on the validity of that addition or change, whilst the update (e.g. to show the new name) is sent from the server to the client (and any other client devices which need to know of the change) when it is ready for the users to see the new information. Because that updating information can be sent to all clients at the same time, this approach is particularly

useful in environments (e.g. real time price feeds for trading financial instruments) where all users need to receive updated data at the same time. Because the server can be made to automatically send out updated data to all clients in real time, and the rendering layer can continuously take this updated data and display it appropriately, the data displayed on the client is 'live'.

The client software is also generic in that it is configurable on demand to the configuration parameters applicable to any client device. In particular, the communications layer can use a broad range of different protocols and bearers; the rendering layer can render to multiple device types. All three layers are entirely independent of the kind of network based application the client software is interfacing with.

Further, the database layer can provide an interactive, 'thick client'-like interface for the client device such that sub-sets of the structured data held on the database layer can be selected, displayed, manipulated, altered or supplemented with much lower latency than conventional web applications. (A 'thick client'-like interface is a functionally rich interface common to desktop applications that accepts user input via mouse and/or keyboard to drive functionality without recourse to a server and is capable of implementing process flow (e.g. opening new windows) and business logic (conditional behaviours dependent upon data values, validation).

The structured data may be in a self-describing meta-language, such as XML. The network over which the client and server communicate may be a WAN such as the internet, but equally may be a LAN or indeed a network internal to a single device. Although in the preferred implementation, the three layers (comms/database/renderer) are in effect a unitary, fully integrated system, it is also possible for one or more of the comms layer and renderer layers to be plug-in components to a browser, with the browser including the database layer. The layers need not be discrete and separate, but can be interwoven. Also, each layer can interact directly with either or both of the other two layers – e.g. the rendering

layer submits server calls directly to the communications layer; the communications layer can interact directly with the rendering layer - invoking actions/alerts.

5 The client software may be permanently resident on the device (e.g. part of the PROM on a mobile telephone) or may be remotely deployable. An implementation which is remotely deployable is a Java applet version of the client software: this is described in the Detailed Description section.

10 Prior art applets have generally been custom built for a particular application and would be impossible to reconfigure for any other application without significant modifications to their native code.

15 Generic rendering applets do exist for presenting content (e.g. Macromedia's Shockwave™ and Flash™ Players, Adobe's PDF Viewer, postscript viewers, spreadsheets such as Microsoft Excel™, word processors such as Microsoft Word™, presentation tools such as Microsoft PowerPoint™ or even web browsers such as Netscape Navigator™ and Microsoft Internet Explorer™) and though they may contain certain similar components to this invention, in each case, the loose coupling of these components means that user interactivity and live updates necessitate the cost of programming the interface using some form of scripting language (e.g. JavaScript™). In most cases, the content is created in a content creation tool (e.g. a word processor or drawing tool), but interactions have to be scripted separately in that tool or by other means.

25 Another class of generic rendering applets (e.g. X-Windows clients, Citrix's MetaFrame™, Microsoft Terminal Server and AT&T's VNC™) are able to provide a rich and interactive interface to a user over a network, though in each case the rendering is controlled by a server that sends low-level display instructions to the client applet as needed (e.g. when a user types in some information, the server receives an event for each keypress and responds with instructions on what text, its location and font to draw on a screen). These applets require

that an application be programmed to run on the server. The server will in some cases (e.g. Citrix and Microsoft Terminal Server) automatically determine the commands to send to the user interface. In other cases (e.g. X-Windows), the server application has to be programmed with explicit instructions for sending rendering information to the client.

5

A further class of applets are those created to work with relational databases (e.g. Oracle SQL*Forms, Microsoft Access™, Powersoft Powerbuilder™, Gupta SQL*Windows™, Microsoft Visual Basic™). In each case, though it is possible to build 'applications' that automatically enable some user viewing and interaction of data in the database, dynamically configurable and real-time rendering of the user interface requires some form of programming or scripting. For the most part, the database is held on the server, so interactions by the user that require complex queries (e.g. filtering) can often result in each client requesting new data from the server database thus increasing network traffic and server load. Furthermore, even in the case where the network communication, the database, and the rendering process are all on the client, they are typically loosely coupled and are deployed as independent components rather than as a single, tightly integrated architecture, and require separate configuration of each component on each client machine.

In most cases, the above examples require device specific client-side installation and configuration. Implementations of the present invention can be deployed identically to disparate devices and configured entirely from the server, even to the extent of being deployed as needed at run time and disappearing when no longer needed.

The thick client performance provided by the local applet is derived in part from distributing some of the data processing and run-time integration overhead (needed to operate web applications etc. rapidly) away from the originating web server/back end application etc. and sharing it instead between the generic applet on the client side (as defined above) and a new server side intermediary, which, as noted above, we refer to as a 'Presentation Server'. (A 'Presentation Server' is generally any kind of server that determines how data will be

displayed on a client device). Because the present invention relates principally to a client server architecture, it may be helpful to describe the entire client/server architecture at this point so that the 3 Layer System can be seen in context. (The 3 Layer System could in principle also be implemented in a peer to peer architecture, or in an entirely closed device, but the client server architecture is currently seen as the most commercially significant).

More specifically, the database level of the 3 Layer System comprises data sent over a network from a remote Presentation Server which in turn receives that data (or super-sets of it) over a network from an originating source, which may be a web server, a back end application, a messages queue or a web service (or multiple sources of these types) able to generate structured data (e.g. XML or SQL format data directly (or indirectly through a generator).

Hence, this client server architecture envisages a new approach which is neither entirely thick client, nor entirely client-server, nor entirely fully distributed, but blends aspects of each. It involves sending the structured data, which the client device needs to manipulate, once only from the originating source (web server etc.) to an intermediary Presentation Server. The Presentation Server then caches and sends this structured data, to the 3 Layer System on the client itself and subsequently keeps the database layer in the 3 Layer System automatically updated (or 'synchronised') with the originating source. The Presentation Server insulates the originating source from needing to know which clients need to be updated, what updates they need, how to communicate with clients, and the user interface at the clients. This overhead is instead handled by the Presentation Server; this approach allows an networked application developer to concentrate on application business logic development, rather than the front-end display/communications and synchronisation elements that traditionally take over 50% of developers' time. By pre-fabricating these complex functions and capabilities into a generic Presentation Server, networked application developers can produce finished applications far more quickly than previously. Scaling to large numbers of clients (1000s) is also possible, even for legacy back end systems which are limited to serving 10 or fewer

clients. This is because the back end system needs to supply information just once to the Presentation Server; the Presentation Server itself is fully scalable (e.g. multiple CPUs; multiple CPUs on multiple machines etc.) to serve the required numbers of client devices.

- 5 There are two modes of operation that can be used together in the same configuration to handle updating. First, server based data updates – where all data interactions/functions are directed to the Presentation Server and then passed to the backend application. Updates are then picked up by and distributed to all relevant client connections. The result of the action (data changes) are then reflected in the local data and rendered.

10

Secondly, local data updates – data interactions are recorded locally in the database layer. Dependent upon the configuration, the data would then be sent to the PS/backend application. Any consequential changes which can be computed locally by the 3 Layer System are performed and the database layer updated, causing the display to be updated.

- 15 Because the computation is entirely local, it is very rapid and gives the impression that the user is interacting with live data, i.e. data which directly reflects values held at the originating source. The presence of the intermediary middleware Presentation Server is not discernible.

- With local data updates, changes to the local database can be also automatically sent by the 3
20 Layer System to the Presentation Server, which in turn automatically updates the originating source. Any consequential changes to data at the originating source are then sent back up to the Presentation Server, which in turn works out which client devices need to know about this update and then sends them out to all client devices affected by the change. Careful design and deployment of web applications and 3 layer systems can maximise the extent of
25 local computations (i.e. performed on the client device by the 3 layer system) and therefore minimise the frequency of interaction with the Presentation Server or originating source, which are typically co-located in a physically remote setting far from client devices and accessible over only over a WAN such as the internet.

In a second aspect of the invention, there is a networked application which, when invoked or run, causes client software as defined above to be run on a client device.

5 In a third aspect of the invention, there is web server which hosts a web application which requires client software as defined above to be run on a client device.

In a fourth aspect of the invention, there is a client device when programmed with client software as defined above.

10

15

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be described with reference to the accompanying drawings, in which:

5

Figure 1 is a schematic of the main components of the AltioLive Client (ALC);

Figure 2 is a schematic showing the order of events, which occur when a client communication is initiated;

10

Figure 3 is a schematic showing the chain of events, which occur when a user does something, which requires additional data;

Figure 4 is a schematic of various aspects of the ALC configuration;

15

Figure 5 is a schematic of a simple Synchronization Engine configuration;

Figure 6 is a schematic of a complex Synchronization Engine configuration;

20

Figure 7 is a schematic of the AltioLive Deployment Architecture;

Figure 8 is a representation of a screen view. The initial data served by the Synchronization Engine is shown in this example;

25

Figure 9 is a representation of a Service Function;

Figure 10 is a representation of a screen view depicting the Datapool;

Figure 11 is a representation of a screen view of a "NEW_SALE" Service Function;

Figure 12 is a representation of a screen view of a "NEW_BID" Service Function;

Figure 13 is a representation of a "Select Product" list, including a GET_APPIMG image
5 server command;

Figure 14 is a schematic of the full system architecture of an AltioLive application;

Figure 15 is a schematic of the interaction between the Synchronization Engine and the
10 Application with emphasis on three main areas; the Data Service Function, Image Service
Functions and Datapools;

Figure 16 is a schematic of AltioLive in Dynamic Mode. In Dynamic mode it is necessary to
change the HTML to reflect the connection to the Synchronization Engine;

15 **Figure 17** is a schematic of the Logon process used by the AltioLive Demonstration
applications.

DETAILED DESCRIPTION OF THE PREFERRED IMPLEMENTATION

The invention will be described with reference to an implementation called AltioLive, from Altio Inc. of Boston, Mass., USA. This Detailed Description is organized as follows:

5

- A. AltioLive – a Brief Overview of the Overall Architecture
- B. AltioLive Client – Overview
- C. AltioLive Presentation Server – Overview
- D. AltioLive Client – Details
- 10 E. AltioLive Synchronisation Engine (SE) – Details
- F. Deploying AltioLive
- G. Security Aspects of AltioLive
- H. AltioLive – Benefits of using AltioLive
- I. Glossary

15

Appendix I “Getting Started with AltioLive” from Altio Inc.; Chapter 3 titled ‘The AltioLive Presentation Server’

Appendix II “Integrating AltioLive” from Altio Limited; Volume 4 Chapters 1- 5 of the AltioLive User Documentation.

20

A. AltioLive – a Brief Overview of the Overall Architecture

AltioLive solves the technical challenges of streamlining development, deployment and
25 maintenance of real-time interactive Internet/web applications. Updates to data items are received ‘live’ at a client device and can be transmitted concurrently and automatically to client devices to avoid users requesting updates and then having to wait for server responses. Access to the application through alternate devices (e.g. wireless access) presents the user with their data automatically rendered for display and navigation on the device, whether it is

a cell phone or a Personal Digital Assistant (PDA). For certain PDAs, AltioLive can be implemented directly (e.g. IPAQ). For cell phones, Web TV and lightweight PDAs, the application is provided via HTML or WML pages generated by a renderer engine in the Presentation Server. Thick client functionality is not provided on these devices.

5

Applications developed with AltioLive provide users the ability to move between windows inside the browser, scroll or sort through lists, resize columns and see details on selected items (including charts and graphs). The windows can be minimized, maximized, moved around and resized, just like in a standard desktop application. Customizable toolbar buttons
10 allow the user to pop-up additional windows to view further details such as lists, forms or graphs. Data entry is provided through the typical controls such as text boxes, checkbox, drop-down lists etc. with additional complex controls such as graphs, tree-views, date and color pickers.

15 Since developing applications with AltioLive requires virtually no coding, development and ongoing maintenance efforts are reduced up to 75% while bandwidth and server-side processing decrease by a similar order of magnitude.

In AltioLive, an Altio Presentation Server ("APS") deploys and intelligently manages real-time, interactive Internet or Web applications by serving appropriate XML data to client
20 devices. The functions of the APS are (a) to receive and store XML data from the source of that data (e.g. web server, web service provider etc.); (b) to supply to each client device an appropriately configured 3 Layer System - in this instance a generic applet (as described above) - this generic applet handles client device display and communications functions; (c)
25 configuration, initial data and supporting files (images and 'skins') and (d) to provide XML formatted data updates to the client in real-time and (e) to process client device requests to process updates and /or provide further information.

The APS therefore assumes responsibility for properly configuring the generic applet so that the database layer (an XML database layer in the case of AltioLive) on the client displays the correct data for a particular user, in the correct format and with window controls appropriate to that user, and in an optimal manner for the display capabilities of different client devices.

- 5 It also assumes responsibility for all communications functions to/from the client device by appropriately configuring the generic applet, which it sends to the client device. The communications functions enable data to be exchanged with any device type and OS (e.g. PC, Apple™, Unix™, Symbian™, Palm™ etc) and over any bearer/protocol.
- 10 The updating function ensures that all client devices are up to date or synchronized. In AltioLive, this function is achieved using one or more clustered 'Synchronization Engines' (SE). A SE is a server-side software application, which runs on the Altio Presentation Server, and coordinates and controls the interaction between multiple clients, Web applications and server data sources over multiple communication protocol types, while still being
- 15 configurable via industry standard XML. The SE allows for enterprise scale real-time data updates without the need for a client to use a browser 'refresh' button; live data synchronization across multiple users is possible. Static applications can therefore be transformed into dynamic ones and these applications can be extended to multiple devices in different formats, all from a centrally managed location.
- 20 In AltioLive, the generic applet is called the AltioLive Client; this applet enables live data to be displayed in fully interactive windows inside a user's browser. Upon a request to the web server or other ultimate data source, by the client device, the AltioLive Client is deployed to the user's machine from the SE as a small Java applet. The applet is generic and self-
- 25 contained – i.e. no additional code is needed to make it suitable for a specific purpose. Instead, it interprets the configuration file sent from the SE to implement different views and behaviours so that it works for a specific web application/client device/bearer combination.

B. AltioLive Client – Overview

The AltioLive client has three layers:

- 5 (a) a generic communications functions layer (establishing and maintaining connections; re-connecting; updating etc.);
- (b) a generic rendering/graphics layer (generates appropriate screen responses to mouse drags and actions etc.) and
- (c) a database layer (stores applet configuration parameters and UI parameters like windows,
10 window controls, data binding to these controls and event models; stores also the web application related data (a super-set of what is displayed at any time)).

All three layers are necessary parts of the 3 Layer System, and each plays an important role in delivering the desired functionality/behavior.

15

B.1. The Communications Layer

The communications layer provides a means to robustly send and receive messages between the network based application running on the client, and the necessary supporting
20 Presentation Server.

Key features:

- Establish connection with Presentation Server to obtain interface configuration information (in a current implementation this is delivered as XML)
- 25 - Establish occasional on-demand connection with the Presentation Server to obtain ad-hoc information as needed by the network based application (e.g. a response to a search query)

- Establish occasional on-demand connection with the Presentation Server to deliver new and/or updated network based application data to the Presentation Server (e.g. when a user submits a form), and await a response on success/failure
- Establish regular polling connections to receive network based application data to be presented in the interface
- Establish and maintain a persistent connection to a Presentation Server if required for low-latency receipt of streaming data
- Establish occasional connections with the Presentation Server to ensure server is alive (functioning) if needed when no streaming data has been received for a significant period.
- Interface to the Database Layer to deliver new and updated information from the Presentation Server
- Interface with Database Layer to receive messages to be sent to the Presentation Server (e.g. when the network based application data is changed)

B.2 The Database Layer

The Database Layer provides a means to hold and query network based application data and user interface configuration data as well as an interface to the Communications Layer and the Rendering Layer.

Key Features:

- In the generic case it receives digital information from the communications layer and efficiently store it for later access on demand
- Receive structured data from the Communications Layer that may later be queried using a query language to combine and/or extract specific subsets of data on demand
- The structured data may be XML; XML querying approaches can be used to combine and/or extract data subsets
- The Database Layer can provide temporary cacheing of network based application data on behalf of the rendering layer, in order to reduce and/or minimize the need

for the 3 Layer System to communicate with the server in the course of running the network based application.

- 5 - The Database Layer supports dynamic additions, modifications and deletions to the information it holds, triggered either by the Communications Layer or the Rendering Layer.
- The Database Layer responds on demand to requests for information from the Rendering Layer.
- The Database Layer may in response to a request from the Rendering Layer, issue a request in turn to the Communications Layer to retrieve further information from
10 the Presentation Server (e.g. when the Rendering Layer has asked for data not yet present in the Database Layer)
- The Database Layer may have a means to manage its size and may choose to expire and/or archive old or infrequently accessed information
- 15 - The Database Layer works in conjunction with the Rendering Layer to automatically trigger changes to the user interface to reflect the current state of the data in the Database Layer

B.3 The Rendering Layer

20 The Rendering Layer provides a means to (a) combine information describing an interface for a network based application with information to be rendered within that interface and (b) dynamically construct and manage the display of the application interface on the host client platform (e.g. a web browser or handheld device).

- The Rendering Layer generates a user interface that reflects the state of the information in the Database Layer
- 25 - Subject to the capabilities of the client device, the Rendering Layer can generate a dynamically updated user interface that continually reflects the latest state of the information in the Database Layer
- The Rendering Layer may support interaction with an end user through inputs from an external input device (e.g. a keyboard, button panel or mouse)

- The Rendering Layer may generate a network based application interface at the pixel level
- The Rendering Layer is responsible for generating and managing user interface controls (e.g. text entry boxes, charts, scroll-bars)
- 5 - The Rendering Layer may display information in a traditional desktop format using multiple overlapping views.
- The Rendering Layer may support user interaction with the user interface such as drag & drop of information between views, entering data into a text box, pressing a button.
- 10 - The Rendering Layer may support rendering of network based applications on devices and/or in a form other than pixels on a screen. (e.g. lines of text on a cell phone or LEDs on a domestic appliance)
- The Rendering Layer may support interfaces to alternative rendering mechanisms independently of its own drawing capabilities (e.g. JavaScript calls and triggers to and
15 from the Rendering Layer)

Other key features of the AltioLive Client are:

- 20 • The generic applet provides a generic display capability by being able to download from the remote Presentation Server one or more of the following, each configured appropriately for the destination browser, device type and bearer: windows, controls on the windows, data binding to the controls and event models. The XML data is bound to the controls and event models, so that a web application developer need only define
25 what data is to be associated with each window control; the applet then handles how data should be displayed when different events occur (e.g. if the data is a list of names, the applet handles how the name list is displayed if a 'sort by alphabetical order' control is selected). This 'View' definition can be made specific to each user – allowing different

users to be able to see or manipulate different data sub-sets and hence providing an access control mechanism.

- 5 • The applet provides a zero-client footprint in that it is (a) installed on a client device only in response to the client device initiating or requesting a web application or web service and therefore does not need to be pre-loaded or installed and, (b) after the client device has ceased using the web application or web service, it may be removed entirely from the client device.
- 10 • The small size of the applet allows it to be rapidly downloaded. Once the Java applet is running on the user's machine, it maintains a persistent connection with the Presentation Server. The Presentation Server will then 'push' or propagate new data to all parties that require it. Only data, which has actually changed, is pushed across the network, reducing the bandwidth used, unlike HTML systems that send an entire page when only a single
15 data point in the page alters. Another alternative to the persistent streamed connection is a polling connection, which makes regular requests for data changes. This uses a timestamp mechanism to ensure that only new updates for the individual client making the request are returned.
- 20 • The XML database provided by the applet enables XML data from two or more different web applications, web services or data sources to be combined or manipulated in one or more windows running in a browser on the client device as though from a single source.
- 25 • The generic applet can generate multiple windows in a browser window of the client device, with a user able to sort data, edit data, or scroll through data in each of the windows, without the need for there to be any client side software development.

- The generic applet allows XML to be cut and pasted into a clipboard or other form of temporary memory. Hence, a user can drag data (e.g. numerics, text or images) from one window generated by the XML database and drop it into another window – the appropriate changes (including computations) resulting from pasting in the new data into the applicable field of the new window are generated and displayed.
5
- The generic applet can generate multiple windows in a browser window of the client device, with each user able to define a personalised arrangement of windows and data to be displayed in each window, with the data to be displayed in each window capable of being obtained from several different web applications or web services.
10
- XML data is associated with pre-defined controls by a developer and then at run time the generic applet automatically displays the data in conjunction with some or all of those controls.
15
- The XML database can be accessed using XPath queries, in which standard XPath queries are enhanced with the following features: indexed retrieval; direct reference to elements and attributes identified by controls.
- Session objects can allow the remote Presentation Server to provide to the XML database the applicable data and configuration parameters appropriate to the client device and the last state of the client device known to the remote Presentation Server . This prevents unauthorised access without a session object. Session objects are user specific, so that users can access a web application irrespective of client device being used. The PS also maintains the notion of a Connection, allowing a single session to support multiple connections (two browser windows for example) each running a different view/configuration.
20
25

C. AltioLive Presentation Server (APS) - Overview

The AltioLive Presentation Server (APS) integrates with Web Applications to efficiently and securely distribute live XML data to users across the Internet through the range of Altio

5 Clients. The APS has the following three functional aspects:

- 10 ▪ Display - The APS provides a trusted architecture for the presentation of real-time interactive applications. Through a tiny, browser-based applet and XML-configuration, Internet applications deliver a completely customisable, dynamic interface with desktop-style functionality (e.g. drag and drop, resizable windows, customisable screen colours, customisable fonts, full UI configurability according to user's role and preferences).
- 15 ▪ Communications - The connection between the application (on the server) and the end-user (on the client) governing the form and method of communication between the application (on the server) and display to the user (on the client). The APS provides a persistent, secure connection between the application and clients, transmitting data via configurable HTTP/HTTPS polling or streaming access mechanisms.
- 20 ▪ Data Synchronization - Managing data distribution across the user-base, providing all users with the latest data, in real time – from any device.

The live data is displayed in interactive windows inside the users' browsers through the
25 appropriate client:

- **AltioLive Client (ALC)** – delivers a live, interactive, thick-client style user interface within the user's browser.

• **Pervasive Client (APC)** – delivers a page-oriented user-interface rendered for user access through simple devices such as PDAs and mobiles.

5 • **Data Client (ADC)** – delivers live, interactive data for presentation through a DHTML interface.

Synchronization Engine (SE)

The APS is implemented as one or more clustered Synchronization Engines (SE)s that are monitored and managed through the SE Admin Tool.

10

D. AltioLive Client (ALC) - Details

15 The AltioLive Client is a computer user interface that radically enhances how information can be displayed and updated over Internet technology.

An ALC is composed of a number of components:

- 20 ▪ An internet client machine;
- A web browser supporting a Java virtual machine;
- A Java applet;
- An XML Desktop Database.

These components are further described below and are shown in **Figure 1**

25

The ALC 100 typically fits into a configuration that consists of a Web browser 103 in which the ALC runs as a small machine independent client applet. The web browser would be running on an Internet Client 104 that would be connected to a web server 105 via the Internet 102. The web server would be the gateway to the Internet for a number of server

based web applications 106a and 106b etc. The Internet Client could be a personal computer running browser software, a Wireless Application Protocol (WAP) mobile phone, a Personal Information Manager (PIM), or other information-viewing device. For live connectivity these devices must support some form of network connection (e.g. by using the
5 Java 1.1 Virtual Machine). The personal computer would typically be a workstation (e.g. PC compatible machine running a Windows™ operating system, Macintosh™ or Unix™ workstation); the browser software could for instance be Microsoft Internet Explorer™ version 5 (See www.Microsoft.com) or Netscape Navigator™ version 4 (See www.netscape.com).

10

In general the web server 105 would control the communication with the client using standard HTTP 109. Typically the ALC would use HTTP for the initial communication but could then switch to using alternative communication protocols, such as TCP/IP or UDP, which better suit the application. In this way the ALC and the SE set up their own
15 communication link 110.

One of the unique features of the ALC is that the Java Applet is small enough to be downloaded and configured on the browser each time the user starts a session with the application. This then gives rich client functionality without the need for add-ins and
20 complex installation procedures.

Once the ALC is running on the client it creates an XML Database (inside the ALC space) on the desktop which not only stores all the data immediately required by the client but also has all the configuration information on how the interface should look and what the user
25 preferences are.

ALC - SE Interaction

The key to the uniqueness of the ALC is how it interacts with the SE. The SE is a server-side layer that serves the ALC with the configuration, data and updates.

As can be seen in **Figure 2**, it is always the ALC that initiates the communication by requesting a page from the web server that contains the ALC applet 201.

5 The web server will pass the request on to the SE, which will return a page to the client containing the ALC applet. The ALC is then initiated in the client browser and from then on establishes and maintains communications with the SE. To allow the user interface to be personalised, the user must make themselves known to the system and will therefore be allocated a unique key that can be used to reference personal preferences. Upon request from the ALC the SE amalgamates XML from a number of sources to send back to the
10 ALC 204. Typically the SE could get information from the following:

- General ALC configuration XML;
- User specific interface preferences;
- Web Application data, from potentially many applications.

15

The SE can be configured to retrieve this information in a number of different ways, including standard HTTP calls, SOAP, JDBC, Socket, or JMS. The ALC will use this data to render a display to the user 205. The user can then interact with the data in a familiar desktop environment. As the user interacts with the application the ALC may contact the
20 SE for additional data or it may initiate an update of the data.

When a user does something that requires additional data a chain of events takes place as described in **Figure 3**. The important step is that the SE can be configured to be an intelligent cache of web application data 304, thereby reducing the load on the web
25 application itself. An example of this would be a news server. If the SE already has the requested news article then it doesn't need to interrogate the web application. The SE can also be configured to only return changed data therefore reducing the amount of data sent 305. The SE uses a timestamp mechanism to ensure that a client connection only receives "new" data changes.

ALC Configuration

The ALC is totally configured with the use of industry standard XML. There are a number of aspects to the configuration:

5

- Interface definition
- Data Definition
- User Preference Definition
- Communications Definition

10

Referring to **Figure 4**, the interface definition 401 defines how the application will be rendered by the client in terms of windows, controls, images etc. This is initially defined for the application but then may be modified to take account of the user's specific configuration. The data configuration 402 defines the data to be served to the client. This is the XML definition of data that will be stored in the data section of the XML Desktop database. (The User Preferences 403, allow user specific requirements to be created and stored in the SE. These can then be used the next time the user logs on.

15

The Communication Definition will define how the ALC should interface with the SE in terms of the underlying protocol. Typically this would be HTTP Streaming or Polling.

20

E. AltioLive – Synchronisation Engine (SE) Details

25 The Synchronisation Engine is a software application that can co-ordinate and control the interaction between multiple clients, multiple web applications or server data sources and multiple communication protocol types, while still being configurable via industry standard XML.

Some unique features of the SE are:

- The ability to interface with existing web applications using a variety of protocols, including conventional HTTP, JDBC, SOAP, Socket, and JMS;
- 5 ▪ In built scalability to support large numbers of simultaneous clients and to keep them synchronised with each other and the web application;
- In built fault tolerance through parallel clustering of SEs;
- The ability to plug together multiple SEs and allow them to discover each other and their data sources;
- 10 ▪ The ability to decouple the refresh rates of the web applications and clients;
- The ability to communicate with a number of different client types, for example Internet browser, WAP phone, Personal Information Manager;
- Ability to pull together multiple web services;
- The ability to transport XML data elements to targeted destinations.

15

An SE is composed of a number of components:

- A network server machine;
- A server operating system;
- 20 ▪ A communications protocol stack;
- A Java™ virtual machine;
- The SE Java servlet control programme,

These components are further described below:

25

The SE would be located on a network server machine. Typically there would be a CPU, random access memory, permanent disk storage, ports for communicating with other servers on the local network, a manual input device and a display unit. Once the server was set up it

would generally be administered remotely via the comms ports, in which case the keyboard and VDU might be removed.

5 The server is designed to run an operating system that is stored on permanent disk and is loaded into memory at start-up. The operating system would typically be Windows NT™ or Unix™ or Linux™, but could be any operating system that can support Java™ Servlets. To communicate over the local / wide area network the server would require a communications protocol stack such as TCP/IP.

10 The software to control the SE would also be stored on permanent disk within the server and would also be loaded into memory as required. All the configuration information for the SE controlling software is stored as XML.

Communications

15 A key aspect of the SE is communications. The SE is able to communicate with back-end applications, with front-end clients and with other SEs. It can communicate at a number of different levels and in a number of different modes. At the lowest level the SE controlling software will need to know the network protocol by which it needs to talk to each client and application. For communication to backend applications this could be HTTP, JMS, SOAP,
20 Socket, or JDBC. Communication with the clients will typically be across the Internet and would therefore be over HTTP, but in some scenarios could be over TCP/IP or UDP.

The SE will need to know the network address of each communications partner, for TCP/IP this would be the IP Address or the server name which this then translated into a TCP/IP address.

25

SE Configuration

The SE would be configured from a number of XML configuration definitions. Referring to Figure 4, the server configuration 401 defines what data pools are going to be available and the definitions of the service functions required to interact with the web applications that are

enabled through the SE. The server configuration also defines the architecture of the SE in terms of clustering, security and communications.

5 The data configuration 402 defines the data to be served to the client. This is the XML definition of data that will be stored in each of the data pools. Each data pool is a transaction log, which tracks changes to the data set(s) held in the client.

10 The application configuration 403 defines how the application will be rendered by the client in terms of forms, screen, images etc. This is initially defined for the application but then may be modified to take account of the users specific configuration.

The application default 404 will store system defaults such as refresh rates and other variables. These will be set at start up.

Data Interaction

15 Once communications are established between the SE, clients and web applications, the SE will need to transfer data between them. This can be by a number of methods:

- Push – This is where the SE propagates data to all parties that require it. This is used to keep the clients in sync with web application data changes. Web applications can also be set up to push data to the SE.
- 20 ▪ Pull – This is where the SE polls for the latest data, on a time basis or on user instigation. The client can also pull data from the SE. An example of a “Pull” is how changes to data on the web application are propagated back to the web client via the SE. Typically a web page is only refreshed when the user requests one and when it is refreshed all the data is transferred from the server even if it hasn’t changed. The SE
- 25 introduces a number of alternatives depending on the application requirements. The SE can be configured to check for changed data at different time intervals 404. If the SE detects a change 408 independently of a RE (Rendering Engine/Layer) request 402, then it can either cache the change or push it to all clients who currently

require it. Changes would only be able to be pushed to the RE if a suitable communications protocol was in place such as HTTP Streaming, TCP/IP or UDP.

5 **Simple Configuration**

The first embodiment of this invention, Synchronisation Engine Simple Configuration, is depicted in Figure 5. This figure shows the simplest configuration consisting of one SE server 504 which would be connected to a number of applications 507 and optionally the web server 508. It would also be connected to the Internet 503.

10

Complex Configuration

Figure 6 demonstrates how multiple versions of the SE can be pieced together to form a scalable and fault tolerant network. This can be done without any code changes and with minimal configuration changes and the configuration that is required is all via XML. The SE system can be deployed many times to produce a hugely scaleable and fault tolerant system. In this embodiment there are two main SEs 601 & 602, these would act as dual redundant servers providing a high degree of fault tolerance. Each SE would have independent links to the source applications 604. The SE invention embodies code that will allow these servers to keep themselves fully synchronised 603.

20

This approach provides a large degree of scalability by allowing the SEs to be setup in a snowflake configuration. The two main SEs, 601 & 602 would act as the hub and would then be connected to other SEs 605 & 606 etc rather than directly to clients. Each one of these satellite SEs could then be linked to a large number of clients. The SE invention incorporates the propagation logic required to service this complex snowflake configuration.

25

More SEs could be setup in parallel to provide additional resilience and scalability. Also the snowflake configuration could be extended to have many layers thereby giving near infinite

scalability. Although in the above-described embodiment, the client was based on a web browser, it would be possible to have other types of user interface such as a Wireless Application Protocol (WAP) phone or a Personal Information Manager (PIM). The SE can also be split over many Central Processing Units (CPUs) within the same server or over
5 many servers.

F. Deploying AltioLive

The generic applet must be configured so that the XML database layer on the client displays
10 the correct data for a particular user, in the correct format and with window controls appropriate to that user, and in an optimal manner for the display capabilities of different client devices. The APS has responsibility for configuring the generic applet. This is achieved via the XML Deployment Package (XDP). XDPs contain the definitions for the following AltioLive elements:

15

1. Service Functions – define the data (XML data & related Image data) available to users from the Web Application and the mechanisms available to update the Web Application.

2. Data Pools – define the mechanisms for live distribution of XML data updates to users.

20

3. AltioLive-Views – define the live interactive user-interface delivered through the AltioLive Client. There can be multiple views according to user roles.

4. Pervasive-Views – define the page-rendered user interface delivered through the Pervasive
25 Client. There can be multiple views according to user roles and/or device capability.

5. Data-Views – define the data and access available through the Data Client for a DHTML user interface.

6. Common elements – icons, desktops styles etc. for use across views within the XDP. XDPs can be developed as self-contained entities. Typically a self-contained XDP is used where the views are tied to a specific Web Application e.g. a contract resource application could provide a requirement-view, a resourcing-view and a phone-based timesheet entry-view. Self-contained XDPs are also used in delivering third-party components (e.g. a Chat or e-mail component) that are effectively 'plugged-in' to application-specific XDPs. XDPs can also make use of definitions from other XDPs to define their views. Typically this is used where the views merge XML data from many applications into a specific role-based view (e.g. a salesman sees a cross-system-view combining sales tracking data, customer management data and credit control; the sales director sees a view combining aggregated sales data, personnel monitoring and campaign performance data) or to use elements of a third-party

XDPs are developed within the AltioLive Development Edition (ADE) and then registered with the APS using the SE Admin Tool. This distributes the XDP across all Synchronization Engines in a cluster to make it available to users.

G. Security aspects of AltioLive

AltioLive combines best-of-breed security technologies and practices to ensure a flexible, configurable and comprehensive security architecture for all applications. AltioLive is designed to integrate within existing security frameworks. It takes advantage of existing session management and Secure Socket Layers and provides unique additional security features to offer a robust and highly secure environment for all systems.

User Authentication

In AltioLive, users do not directly log-on to the APS. AltioLive intentionally reuses the session object created in the web server/application server in order to let you take advantage

of:

- existing authentication policies for the application;
- best practices in user authentication;
- 5 ▪ and the existing investment in technology infrastructure

When the end-user accesses the application through a host Web page, this page downloads the appropriate Client from the same web server. The Client then contacts the APS, which uses the existing session object (on the server) to determine the specific View and XML data
10 to be served to the user. Even if the URL of the host Web Page is known, the user cannot activate their connection with the APS unless they already have a valid session (including the necessary AltioLive specific variables) on the Web server. Using the existing session from the Web server also prevents users from having to re-authenticate as they cross boundaries in a distributed system. AltioLive allows users to authenticate only once, regardless of the
15 machine on which the application is deployed. To control who can connect to the APS, AltioLive leverages existing application servers' ability to enforce rules that govern whether a connection should be established based on client IP access or protocol. This allows companies to decide if "Client" refers to AltioLive Client, AltioLive Pervasive Client, and the AltioLive Data Client.

20

Unique User Views and Data Pools

Individual or groups of users can be assigned different views/windows, which can be populated by data specific to each group's or individual user's need. Groups and/or
25 individually registered users can subscribe to different data pools in the APS. Users only see the data to which they subscribe. In this way, AltioLive prevents unauthorized users from seeing sensitive data as well as allowing for unique user experiences/views. Through the APS, system administrators can control which users see which data to ensure only the right data reaches the right users. For example, views could be configured for different roles in an

organization (i.e. Salesperson, Sales Manager, Credit Controller etc.) that provide access to a number of backend systems – each role would have a view that restricts what functions are available and deliver only data that is relevant to their role. Specifying individual access levels through the APS allows production systems to be up and running very quickly with minimal configuration and no coding.

Firewalls

Use of standard HTTP and HTTPS connections between the Client and the APS makes the technology transparent to most firewalls and does not require any relaxation of existing security policies. In highly secure environments where connections are restricted, the Client/APS connection is specific and easily added to the allowable list of sites. An additional option for system administrators is to locate departmental slave (APS) servers that sit inside the firewall to support one or more users (who therefore no longer have to connect to the Internet). The slave server has a dedicated peer connection to its master APS that can be additionally secured according to company security policies

Connection Security

Not only is it important that users are properly authorized and authenticated when accessing data from the APS, but also that this data cannot be viewed while in transit. To avoid eavesdropping on connections between any connections to the APS, AltioLive offers users a number of security features:

25 Client to/from APS

AltioLive includes support for a Secure Sockets Layer (SSL) and digital certificate implementation that provide authentication and privacy via encryption in distributed APS applications. The Client/APS connection takes advantage of current SSL technology

including 56 and 128 bit cipher strength connections. AltioLive is compatible with certificate authorities that support X.509v3 certificates. This means that developers are free to use the vast majority of certificate authorities available today.

- 5 In secure operation, the APS requires a digital certificate to be able to prove its identity to Clients and other servers. When a Client contacts the APS using SSL, they first request the server's digital certificate to prove its identity. Once the Client receives the server certificate, identity is assured, and the Client and APS can confidently send data back and forth across the network. For highly secure environments, many installations need to also authenticate the Clients accessing the APS. For this reason, the APS SSL installation can be configured to also provide authentication for Clients and Web browser. This is called a two-way authentication and incorporates the same digital certificate verification process. The major difference for two-way authentication is that the digital certificates are located in both the Client and the APSs. Once certificates have been exchanged and mutually verified, both the Client and the APS can trust the identity of the other. Furthermore, once mutual authentication has been set up, a private channel can be created between the two hosts consisting of encrypted data. Many systems will also require that the data which the Client receives from the APS and vice versa to be secure/encrypted. SSL allows both the APS and the Client to encrypt the data exchanged between them. Potential hackers acquiring the data in transit are therefore unable to understand it. Streaming data connections are designed to make efficient use of secure connections to reduce server load as opposed to regenerating cipher keys for every request. Use of compression techniques on the data stream further enhances security while reducing the overhead of encryption on the overall system. Both the Client and the APS monitor connections to detect loss of connection. Clients' connections can be configured to either fail-over to another APS or require the user to re-establish their session through a login.

APS to/from APS

The APS can be clustered for load balancing in master/slave relationships (as well as peer-balanced relationships). These connections are effectively point-to-point connections that can use HTTP, HTTPS or socket connections to secure the interface. If the APSs are distributed in geographically different areas (not in the same data center behind the same physical firewall, for example), the SSL protocol provides secure connections by allowing these APSs to connect over the Internet and authenticate each other's identity and by encrypting the data between the APSs. For security above SSL, hardware encryption can be used on the specific connection between servers. All APSs accessing the cluster must be pre-registered with compatible license keys, which prevent spoofing of any of the servers.

APS to/from backend Web Application

The backend Web Application and the APS are usually in a closed and physically secured environment to prevent direct access to backend data from any outside client device. For implementations where the system connects across the Internet to external Web Services, SSL and digital certificates provide the necessary authentication and data encryption to ensure that the Web Service is a proper entity and all data transmitted is only decrypted and viewed by the intended recipient.

Reliable Data Delivery

Data integrity and delivery is assured through a combination of HTTP generated exceptions and the APS/Client timestamp mechanism. All messages from the APS to the Client are time stamped and sent across the network using HTTP(S). Data from the APS is sent to the Client using HTTP. Should a loss of network connectivity or a temporary hardware failure cause any data not to be received by the Client, the built-in handshaking in the HTTP protocol generates an exception. Essentially, if a message cannot be sent from the APS, the TCP/IP protocol generates an exception. This is generated in the server's Java Virtual Machine (JVM) and the exception is handled by the APS. The APS closes the connection

upon receiving an HTTP exception. The Client will automatically reestablish the connection with the APS (the reconnection strategy is configured through AltioLive Tools). When the connection is re-established, the Client requests all updates since the last timestamp that it successfully received, guaranteeing delivery of all messages in the correct sequence. For
5 streaming data to the browser, the Client is just listening for data updates. In the case where it does not receive any updates or a probe message in a configured interval, it will start its reconnection process to the APS. In the meantime, the APS will generate an exception when the probe message cannot be sent.

10 Transaction Authentication

To prevent an unauthorized user submitting a transaction it is common to use forms of transaction authentication. This takes the form of access security/signature software on the client (sometimes even coupled to physical security – e.g. card swipe, challenge/response
15 devices etc.) can add a security signature to any transaction. The AltioLive Client can be configured to call-out to such applications before submitting transactions to the server to either append the signature or encrypt the whole transaction.

Use of shared workstations

20

Unlike many solutions, the Client does not store any information locally on the workstation (configuration, data or even images). This means that the user can safely access their application from shared/networked platforms without worrying that their data is accessible by others when they have logged-off. This is a key issue in the access-anywhere features of
25 Altio's technology. Standard session time-outs can be used on the server to time-out inactive sessions. The Client can also be configured to disconnect on logout and/or moving from the host page to prevent someone using the Back/Forward functions of the browser to reactivate the application (if it hasn't already timed-out).

Data Available to the User

The data available within the XML Desktop and visible to the user is configured at design time and enforced by the APS. Users cannot configure their Client to access additional data not presented to their specific View.

Protection for the User

For the end-user, the Client is implemented within the standard Java 1.1 Security model. AltioLive Clients therefore do not have access to local data or services on the end-user's system and developers cannot undertake bespoke development to override the functionality of the Client. As an option, the Client can be 'signed' to give end-users additional comfort that the software can be trusted for use on the specific application.

A Complete Security Architecture

In addition to all of AltioLive's security features, companies must also ensure that they secure the general environment in which AltioLive is deployed. Such precautions include:

- Securing the operating system: No security hole in the DNS, sendmail or any other potential entry points.
- Securing File System: No general access to sensitive configuration information
- Securing Network: Use of firewalls and VPN
- Securing Physical Environment of the Server(s): Only authorized personnel have physical access to system

- Separating Development and Production Systems

- No Source Code Available on Production System

- 5 The combination of real desktop-like functionality in online systems that users can access from anywhere, and the Internet becoming a proven and reliable medium for mission-critical communication is now driving companies to rapidly build new systems online as well as migrate existing systems to pure Internet Protocol-based applications using software platforms like AltioLive. Companies deploying IP-based applications must take special care
- 10 that implementations are done in a secure and highly reliable environment to protect the substantial capital commitment required to build or to migrate any application online. AltioLive is designed to take advantage of existing security frameworks, such as session management, user authentication and Secure Socket Layers, reliable data delivery and transaction authentication, and it provides additional unique built-in security functionality to
- 15 offer the most robust and highly secure environments for developers and end-users at a fraction of the cost and time of alternative techniques.

H. Benefits of using AltioLive

- 20 With its XML-based architecture, AltioLive eliminates all the challenges posed by HTML, JavaScript and Dynamic HTML in the development of Internet applications.

Specifically, AltioLive provides companies the following benefits:

- 25 • **Increased User Functionality:** AltioLive provides a client/server-like environment directly inside the browser, giving users the same power as they have in traditional offline client/server-based systems. With AltioLive users have all the data they need at their fingertips in real-time, allowing the functionality in the system to be richly interactive.

Graphs, drag-and drop, sorting, resizing and dynamically changing data are all available to users right inside the browser.

- 5 • **Enable Data delivery Across Devices:** AltioLive allows end-users to access Web applications from any Internet enabled mobile device (including Palm, WinCE, WAP phones, and Web TV) – freeing them from the traditional PC-centric Internet experience. AltioLive applications can be deployed almost anywhere as they are based on pure XML and can be 'tuned' to meet people's various needs.
- 10 • **Savings on System Development:** Using the AltioLive Designer tool, developers drag and drop lists, graphs, buttons, images, and design windows with the use of their mouse. There is no coding required to develop the front-end, and only minimal coding for the connection between the front-end and the back-end. This frees up precious programming resources to focus on building business logic rather than tedious and error-prone HTML.
- 15 AltioLive significantly shortens development cycles, to enable faster time-to-market with considerable cost savings. Companies using AltioLive for large development projects have seen as much as 90% time and cost savings, equivalent to more than a million dollars in upfront cost.
- 20 • **Savings on System Maintenance:** Since there is no code to be written for the front-end by the developers, AltioLive eliminates the chance of bugs through human error. Changes to the front-end can be implemented immediately using the AltioLive Designer tool. Connections from the client to back-end servers are also handled by AltioLive and while some of the integration requires actual code from developers, most of the tasks are
- 25 automated by AltioLive. AltioLive reduces bugs and eliminates the need to have significant IT resources or consultants on hand whenever changes have to be implemented.
- **Savings on Bandwidth:** AltioLive pushes data across the network only when the data actually changes. Even then, it only sends across the data that needs to be updated, not entire

1. The user loads the logon HTML page in the browser, by typing the URL or following a link.
2. The Logon page prompts the user to input user Name, Password and Application name.
- 5 3. When user clicks 'OK', a communication link to the Logon Servlet is established.
4. The Logon Servlet verifies the user Name, Password and Application name.
5. If the logon information is correct, the Logon Servlet creates a Session Object with a unique Session ID. The Session ID is stored on the Application server with the session lifetime. The Logon Servlet also stores the parameters, AL_logon.name and
10 AL_logon.config in the Session Object.
6. The name of the Client applet along with the Session ID (as a cookie) is passed to a Host HTML page.
7. The Host HTML page then invokes the Client. It sends the Session ID and the Logon request to the Sync Engine.
- 15 8. The Sync Engine restores the Session Object with the Session ID. It retrieves the user Name and Application configuration file name from the Session Object. As a part of the Logon request, the user is then subscribed to the appropriate Datapools.
9. The Client now sends an initialisation request to the Sync Engine accompanied by the Session ID.
- 20 10. The Sync Engine requests Initial Data from the Application database.
11. The Sync Engine reads the Application configuration from the Application configuration file.
12. The Application configuration along with user preferences and the Initial Data is sent to the Client in response to the initialization request.
- 25 13. After the initialization process, the Client sends the first Data Update request to the Sync Engine. The Data Updates are either streamed to or polled from the Client. If the streaming mechanism is used, the streaming connection is opened in response to the first Data Update request and new data, if available, is sent to the Client. If the

polling mechanism is used, the first poll occurs after the logon and new data, if available, is sent to the Client.

In the above process, the propagation of the Session ID at each stage is essential, since the
5 Sync Engine can restore the session if it has received the Session ID.

Independent Logon

In your application you may have an existing Logon process or you may wish to write one. This logon mechanism must also create the session. You don't need the AltioLiveDemo
10 servlet and XML files once you log on through your own Logon handler. The Host page needs to be modified accordingly. If you would like an example HTML logon page and the Java servlet that it calls, please contact Altio Technical Support.

If no Logon process is required, then you can directly type the URL of the HTML page that calls a Java servlet. This servlet must perform the following tasks.

- 15
- Creating a session with appropriate parameters
 - Passing the Client applet name along with the Session ID as the cookie to the Host HTML page

When a session is created, the following tags need to be included:

- 20
- logon.name - holds the user name
 - logon.conf - holds the name of the view configuration XML (client)
 - logon.app - holds the APPID as defined in the Sync Engine Admin tool.
 - Further tags specific to your application may also be included.

This means that anyone with an internet connection could conceivably use your application. It also means you won't be able to track which users are using the application.

25 A Technical Note detailing the No Login Handler is available from Altio Tech Support.

CLAIMS

1. Client software, enabling a client device to run a network based application which uses structured data, in which the client software comprises:
 - 5 (a) a communications layer to send and receive messages over the network;
 - (b) a database layer to store, and allow querying of, the structured data;
 - (c) a rendering layer which generates, from the structured data in the database layer, data for a user interface;
- 10 wherein the client software is self-contained to provide all of the communications, data storage/querying and rendering resources needed to run the network based application on the client device.
2. The client software of Claim 1 in which the database layer holds, independently of
15 the structured data, configuration data which defines how that structured data can be interacted with from within the user interface.
3. The client software of Claim 2 in which the rendering layer continuously combines
20 sub-sets of the configuration data and the structured data at display time.
4. The client software of Claim 1 in which the database layer and rendering layer together provide an interactive, thick client interface for the client device such that
25 sub-sets of the structured data held on the database layer can be selected, displayed, manipulated, altered or supplemented.
5. The client software of Claim 2 in which the structured data and the configuration data is set over a network from a presentation server.

6. The client software of Claim 1 which is generic in that it is configurable on demand to the configuration parameters applicable to a client device.
- 5 7. The client software of Claim 6 in which the communications layer can use a broad range of different protocols and bearers, and the rendering layer can render to multiple device types.
8. The client software of Claim 1 in which the structured data is in a self-describing meta-language, such as XML.
- 10 9. The client software of Claim 1 in which one or more of the communications layer, database layer and rendering layer are plug-in components.
- 15 10. The client software of Claim 1 which is implemented as a remotely deployable applet.
11. The client software of Claim 1 which can generate multiple windows in a browser window of the client device, with the data in each window obtained from the database layer.
- 20 12. The client software of Claim 1 which allows different users to be able to see or manipulate different sub-sets of structured data relating to one or more network based applications and hence provides an access control mechanism.
- 25 13. The client software of Claim 1 in which a change in the structured data held in the database layer leads the database layer to automatically communicate with the rendering layer, causing the rendering layer to update the user interface appropriately.
14. The client software of Claim 1 in which a change in the user interface caused by user interaction causes the rendering layer to notify the database layer of the change.

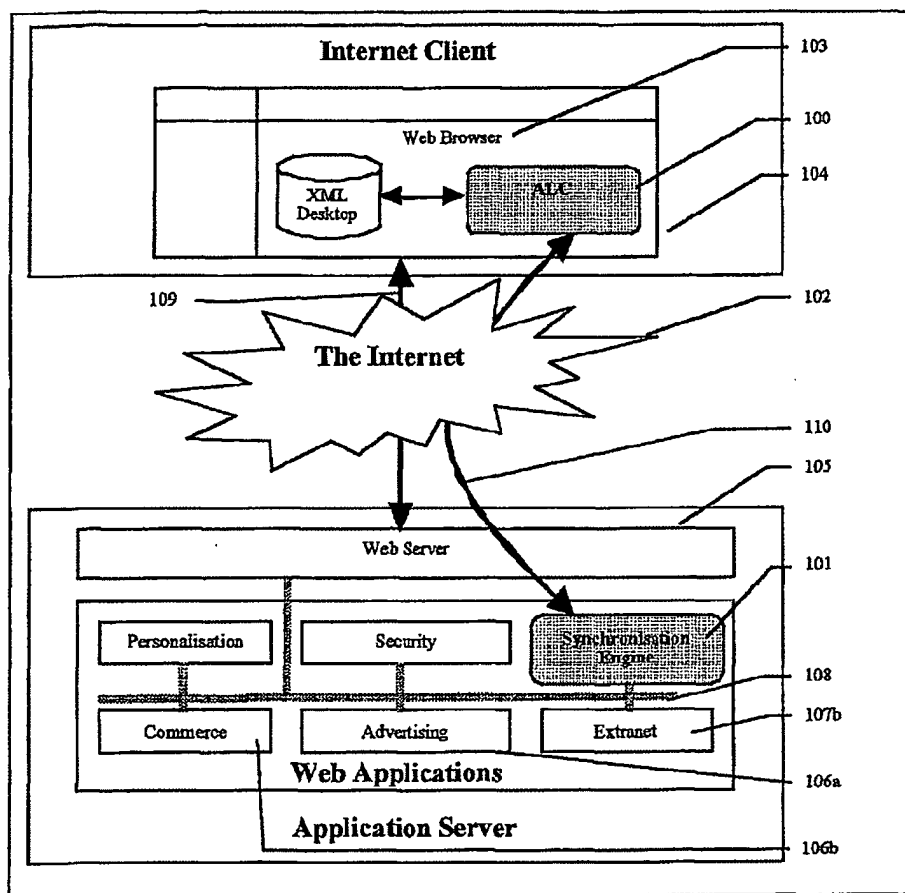
15. The client software of Claim 14 in which a change to the database layer caused by the rendering layer can automatically cause the database layer to contact the communications layer and can cause it in turn to send messages over a network that indicate the change.
16. The client software of Claim 14 in which a change to the database layer caused by the rendering layer does not lead to the database layer contacting the communications layer so that the change is applied only locally at the client device.
17. The client software of Claim 1 which provides a zero-client footprint in that it is (a) installed on the client device only in response to the client device initiating or requesting a network based application and therefore does not need to be pre-loaded or installed and, (b) after the client device has ceased using the network based application, it is removed entirely from the client device.
18. The client software of Claim 1 in which the database layer enables structured data from two or more different network based applications to be combined or manipulated in one or more windows running in a browser on the client device as though from a single network based application.
19. The client software of Claim 1 which can generate multiple windows in a browser window of the client device, with a user able to sort data, edit data, or scroll through data in each of the windows, without the need for there to be any client side software development.
20. The client software of Claim 1 which allows XML structured data to be cut and pasted into a clipboard or other form of temporary memory.

21. The client software of Claim 1 which can generate multiple windows in a browser window of the client device, with any user able to define a personalised arrangement of windows and data to be displayed in each window, with the data to be displayed in each window capable of being obtained from several different network based applications.
22. The client software of Claim 1 in which structured data is associated with pre-defined controls by a developer and then at run time the client software automatically displays the data in conjunction with some or all of those controls.
23. The client software of Claim 1 in which the database layer can be accessed using XPath queries, in which standard XPath queries are enhanced with one or more of the following features: indexed retrieval; direct reference to elements and attributes identified by controls.
24. The client software of Claim 5 in which a session object allows the remote presentation server to provide to the database layer the applicable data and configuration parameters appropriate to the client device and the last state of the client device known to the remote presentation server.
25. A network based application which, when invoked or run, causes client software as defined in any preceding Claims 1 – 24 to be run on a client device.
26. A web server which hosts a network based application which requires client software as defined in any preceding Claims 1 – 24 to be run on a client device.
27. A client device when programmed with client software as defined in any preceding Claims 1 – 24.

1/16

Figure 1

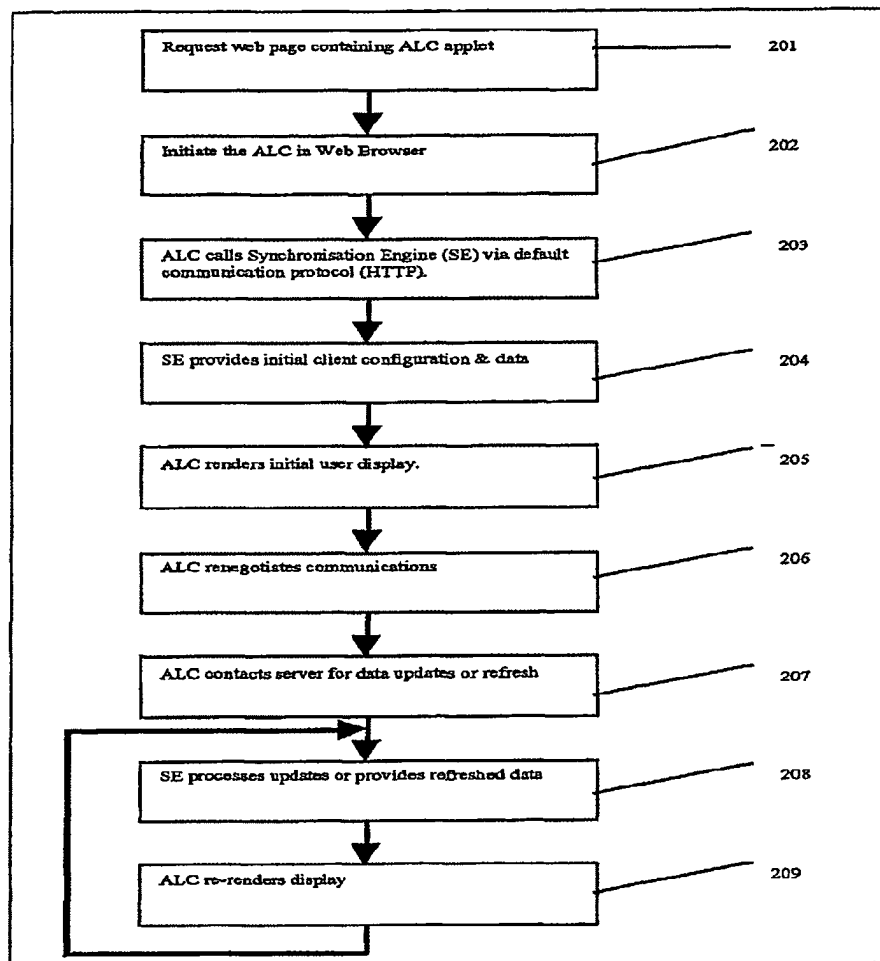
Overview of the components of the ALC



2/16

Figure 2

Initiate Client Communication



3/16

Figure 3

User Requests Data

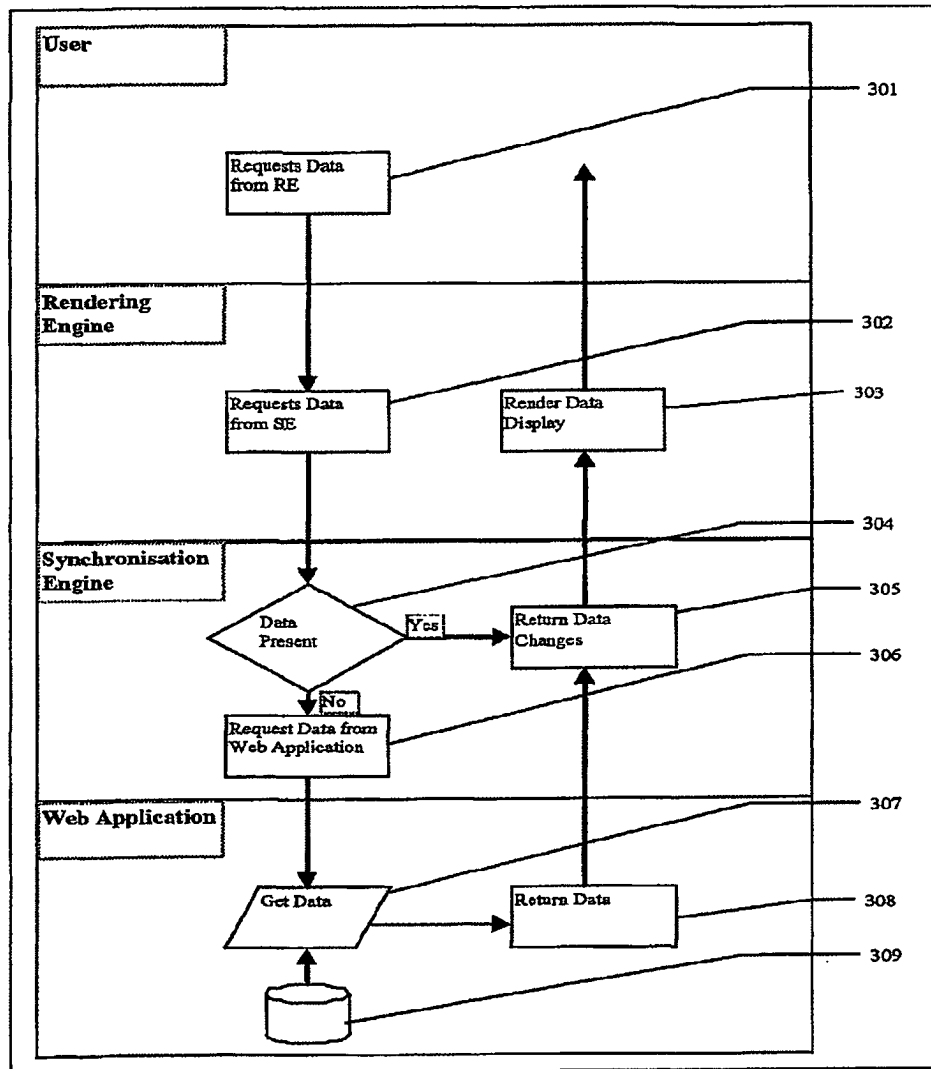
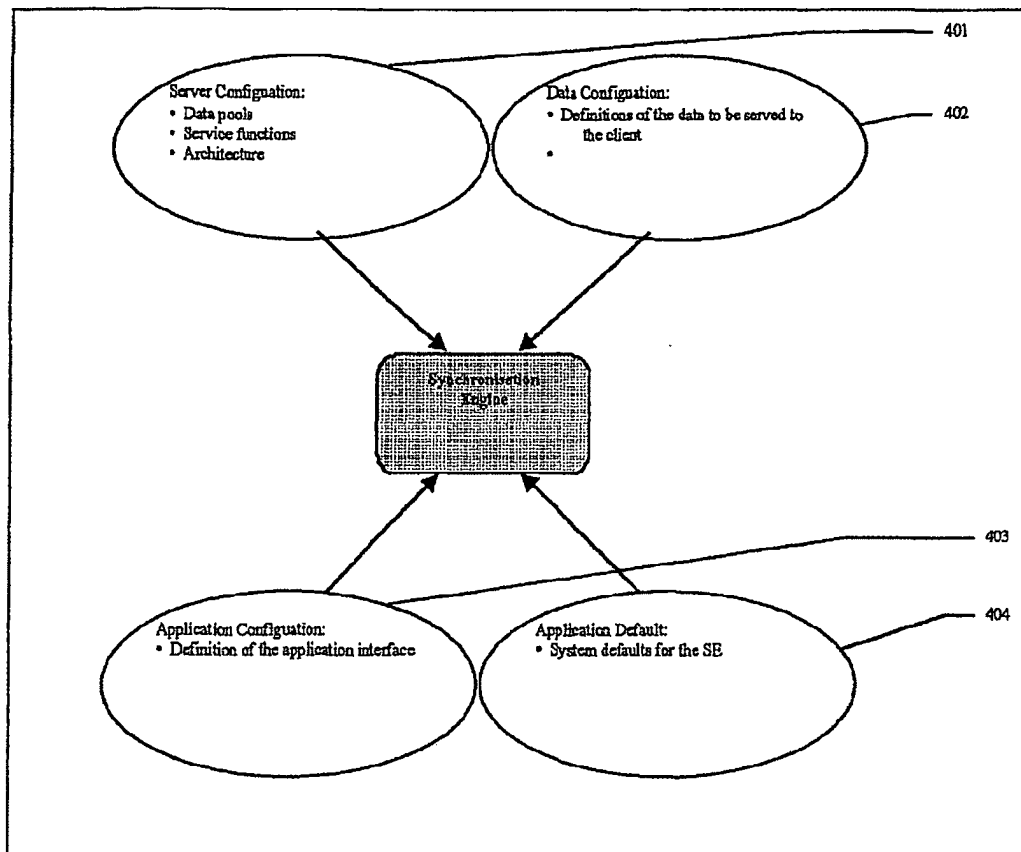


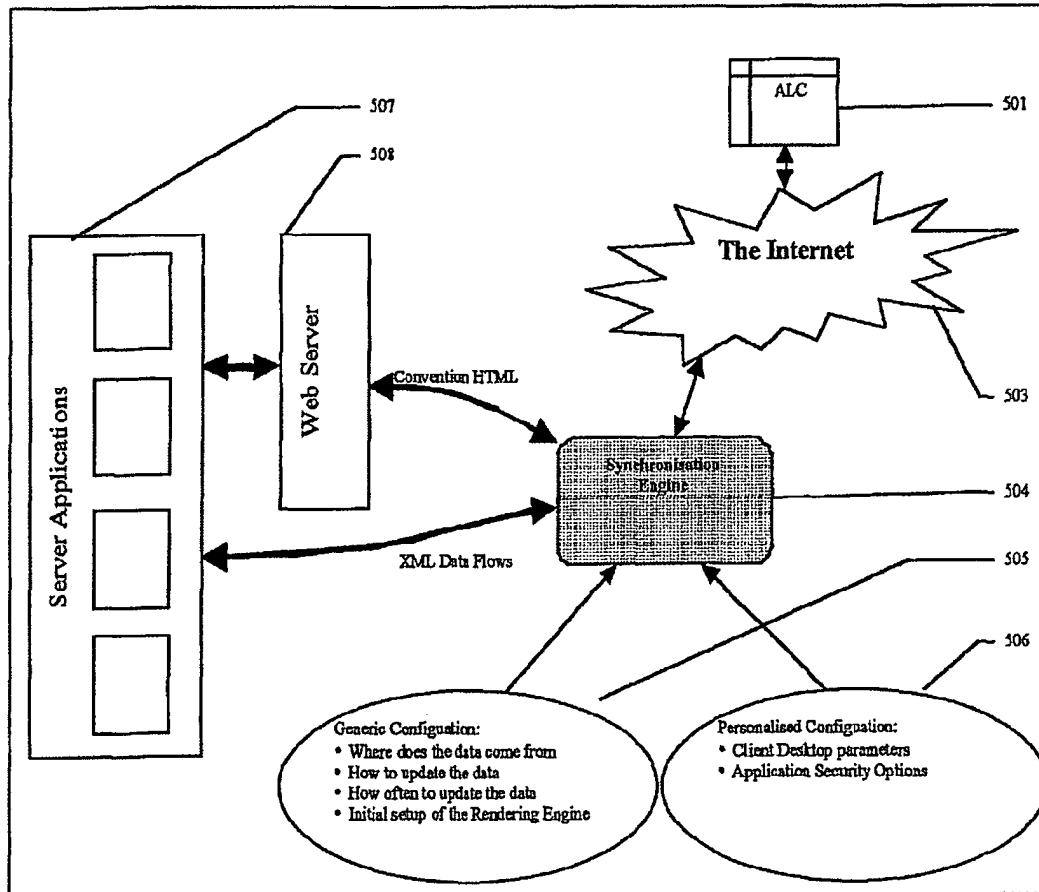
Figure 4
AltioLive Client Configuration



5/16

Figure 5

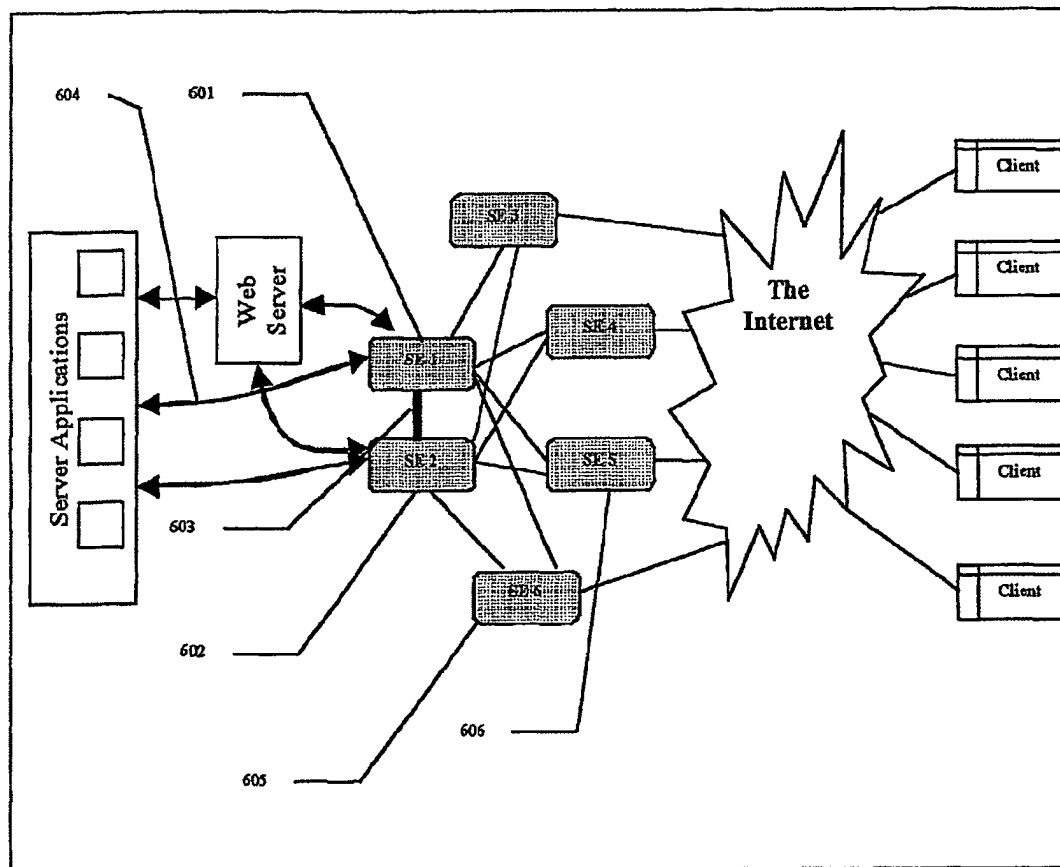
Synchronisation Engine Simple Configuration



6/16

Figure 6

Synchronisation Engine Complex Configuration

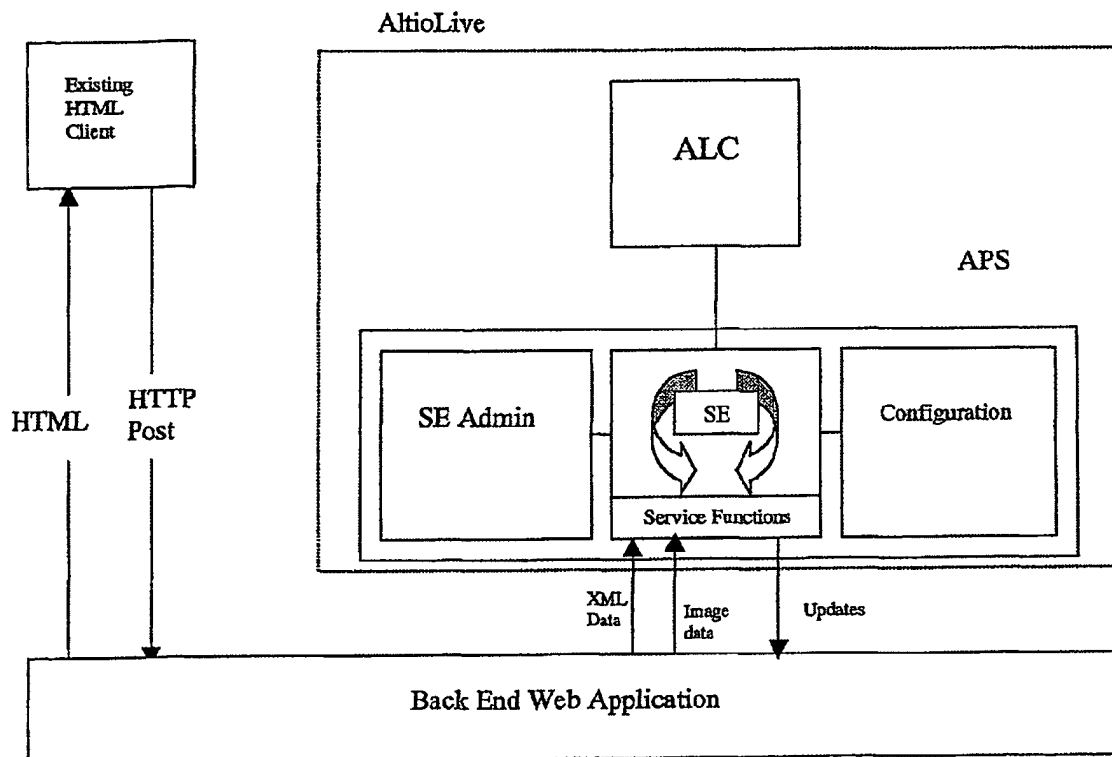


7/16

Figure 7

The AltioLive Deployment Architecture

Architecture



8/16

Figure 8

The Synchronization Engine serves the initial data according to the information associated with the View (shown below).

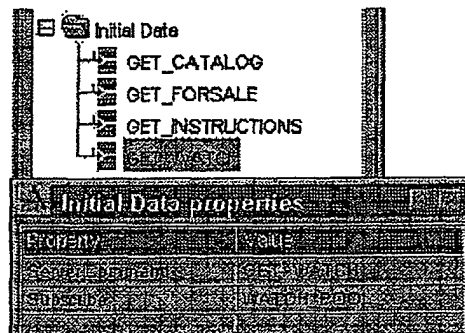
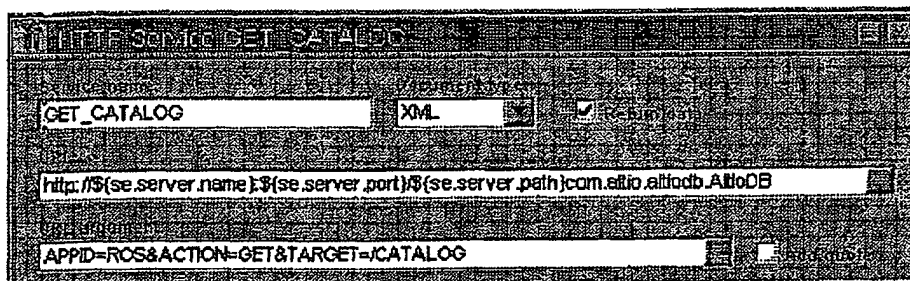


Figure 9

An example Service Function.



9/16

Figure 10

This screen view shows the datapool, which is used to get updates of items which the user has placed on their watch list.

WATCH_POOL 200

WATCH/SALE_ITEM_WI@USR=\${session.logon.n} data/cbflush.bkp

http://\${se.server.name}:\${se.server.port}/\${se.server.path}com.attb. http://\${se.server.name}:\${se.server.port}/\${se.server.path}com.attb.

APPID=ROS&ACTION=GET&TARGET=WATCH APPID=ROS&ACTION=GET&TARGET=WATCH

1 1

WATCH/SALE_ITEM_WI@USR=\${session.logon.name}

10/16

Figure 11

A NEW_SALE Service Function.

NEW SALE

Server: NEW_SALE XML ☐ HTTP

URL: http://\$(se.server.name)\$(se.server.port)/\$(se.server.path)com.attio.attiodb.AttioDB

Action: APPD-ROS&ACTION=NEW&TARGET=FOR_SALE&AL_NAME=SALE_ITEM8 ☒ POST

HEADER ☐ Add new header Success ☒

Header Name: DEFINE Your for-sale item has been accepted

Header Value: DEFINE There was a problem processing your request

Submit Input:

Server	Client	Test
PROD	PROD	
PROD	PROD	
DEV	DEV	
PROD	PROD	
CURRENT	CURRENT	
CURRENT	CURRENT	

Enter Test:

APPID=ROS&ACTION=NEW&TARGET=FOR_SALE&AL_NAME=SALE_ITEM8&GUID=SALE_ID

11/16

Figure 12

A NEW_BID Service Function

NEW BID

XML

☐ SOAP

[http://{se.server.name}/{se.server.port}/{se.server.path}com.allo.allodb.AlioDB](#)

APPID=ROS&ACTION=NEW&TARGET=/FOR_SALE/SALE_ITEM@SALE_ID=

☒

HEADER

DEFINE

Your bid has been accepted

DEFINE

There was a problem processing your request

Server	Client	Test
PROD-27	PROD-27	
PROD-28	PROD-28	
PROD-29	PROD-29	
PROD-30	PROD-30	
PROD-31	PROD-31	
PROD-32	PROD-32	
PROD-33	PROD-33	
PROD-34	PROD-34	
PROD-35	PROD-35	
PROD-36	PROD-36	

APPID=ROS&ACTION=NEW&TARGET=/FOR_SALE/SALE_ITEM@SALE_ID="{SALE_ID}"&AL_NAME=BID&GUID=BI

D_ID

12/16

Figure 13

This element specifies the column of the "Select a product" list, including a GET_APPIMG image server command.

The screenshot shows a configuration window titled "HTTP Service GET_APPIMG". It contains several fields and sections:

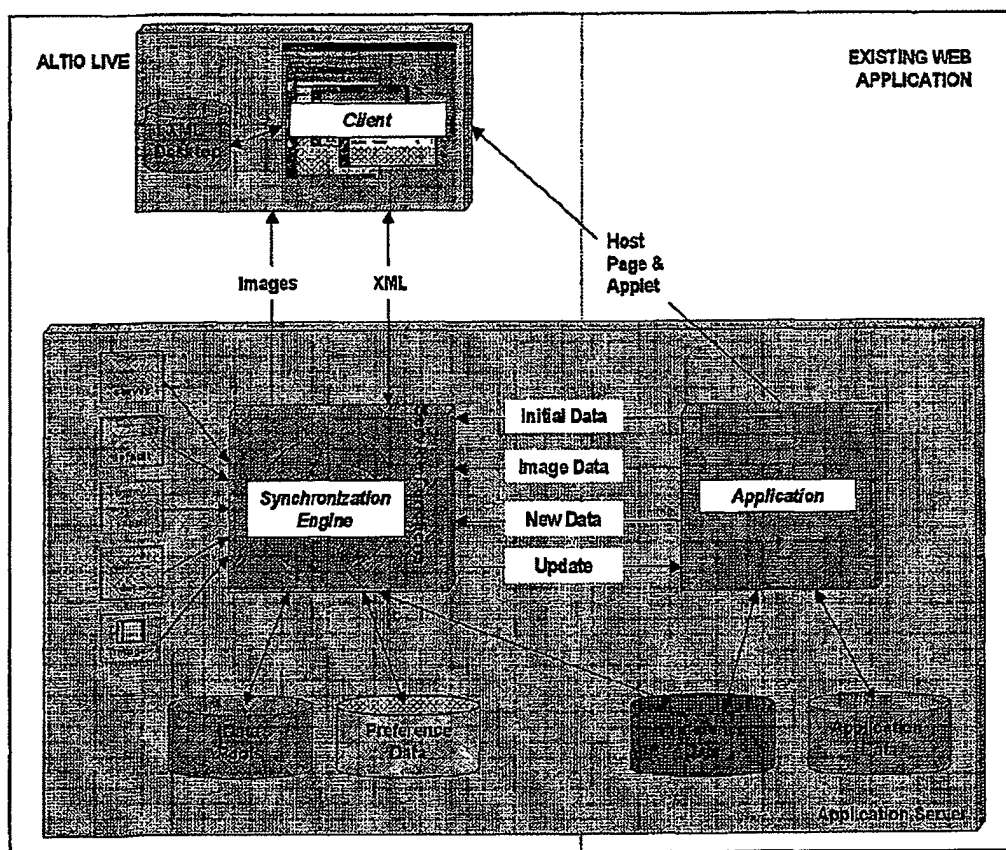
- GET_APPIMG**: A text field containing "GET_APPIMG".
- MO**: A dropdown menu with "MO" selected.
- Region**: A checkbox that is checked.
- URL**: A text field containing the URL: `http://{se.server.name}:{se.server.port}/{se.server.path}com.alio.alio.do.GetImageServlet`.
- APPID=ROS**: A text field containing "APPID=ROS".
- Source**: A dropdown menu with "NONE" selected.
- Destination**: A text field.
- Source**: A dropdown menu with "NONE" selected.
- Destination**: A text field.
- Source**: A dropdown menu with "NONE" selected.
- Destination**: A text field.
- Table**: A table with three columns: "Server", "Client", and "Test".

Server	Client	Test

13/16

Figure 14

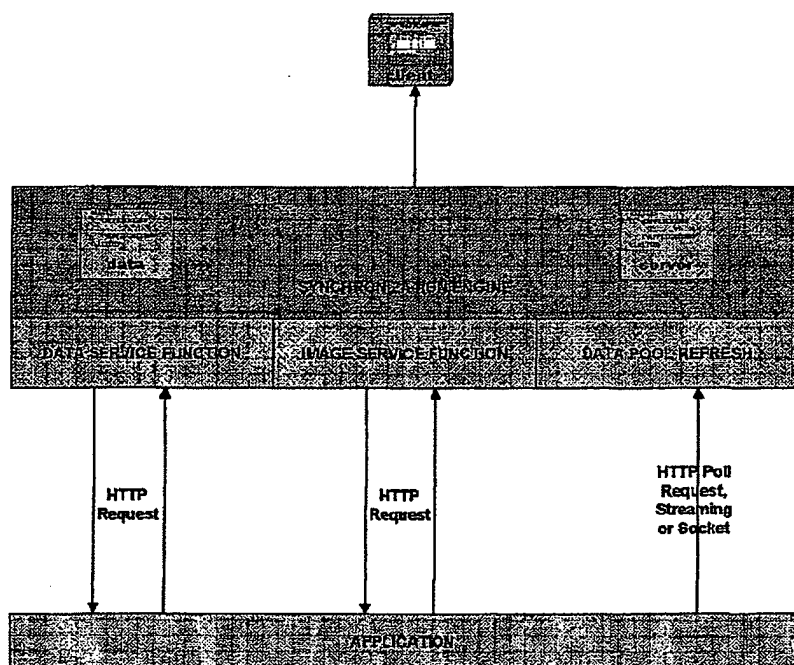
The system architecture of an Altio Live application.







14/16

Figure 15

The interaction between the Sync Engine and the Application.

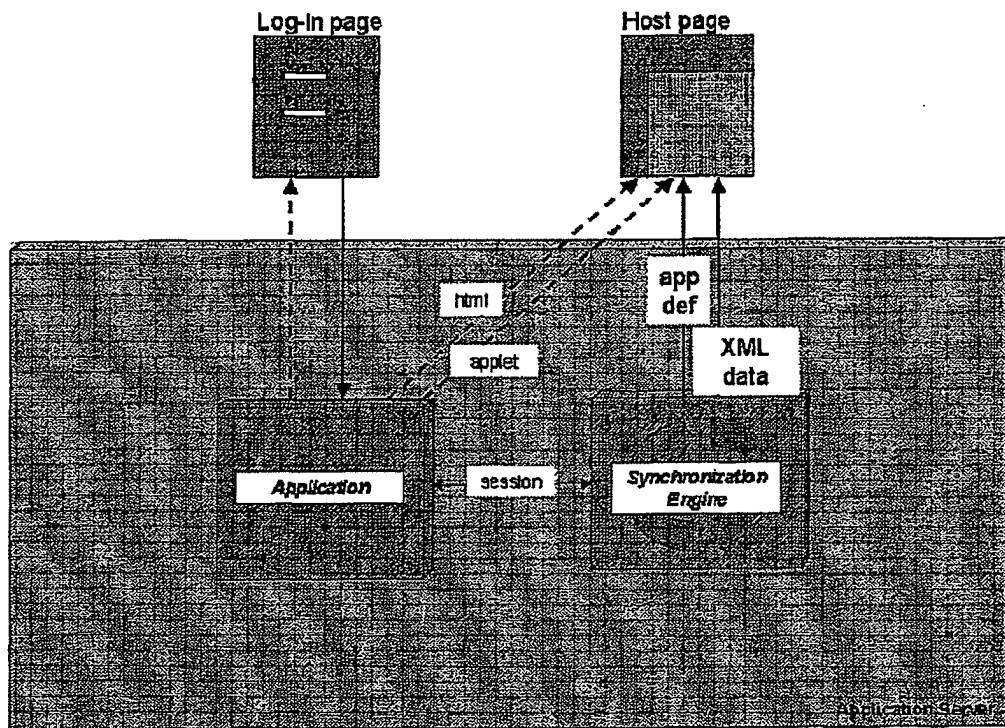
**Key:**

Icon	Name	Notes
	Sync Engine (Sync Engine)	The Altio Live Sync Engine running on an Application Server
	Client	The Altio Live Client running within a browser.
	Server Configuration	XML definition for the Sync Engine (Datapools, Service Functions, Architecture etc)
	Data Configuration	XML definition for the data to be served to the Client.

15/16

Figure 16

When using Altio Live in Dynamic mode it is necessary to change the HTML to reflect the connection to the Sync Engine. Figure 16 shows a typical example of this integration.



16/16

Figure 17

The Logon process used by the Altio Demonstration applications.

